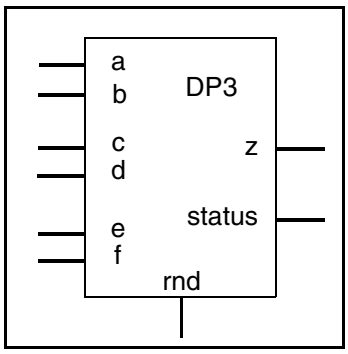
DW_fp_dp3

3-Term Floating-Point Dot-product

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

Revision History



- The precision is controlled by parameters, and covers formats in the IEEE standard
- Exponents can range from 3 to 31 bits
- Fractional part of the floating-point number can range from 2 to 253 bits
- A parameter controls the use of denormal values
- More accurate than using a combination of basic FP operators.
- Provides a variety of rounding modes.
- DesignWare datapath generator employed for reduced delay and area.

Description

DW_fp_dp3 is a floating-point component that computes the dot-product of six floating-point inputs (`a`, `b`, `c`, `d`, `e`, and `f`) to produce a floating-point result `z = a * b + c * d + e * f`, where the symbols (*) and (+) represent floating-point multiplication and floating-point addition, respectively. Therefore, it incorporates three FP multiplications and two FP additions. The accuracy of this component is greater than the accuracy of an implementation using DW_fp_mult components and DW_fp_add components.

Component pins are described in Table 1-1 and configuration parameters are described in Table 1-2.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
a	(*sig_width* + *exp_width* + 1) bits	Input	FP input data
b	(*sig_width* + *exp_width* + 1) bits	Input	FP input data
c	(*sig_width* + *exp_width* + 1) bits	Input	FP input data
d	(*sig_width* + *exp_width* + 1) bits	Input	FP input data
e	(*sig_width* + *exp_width* + 1) bits	Input	FP input data
f	(*sig_width* + *exp_width* + 1) bits	Input	FP input data
rnd	3 bits	Input	Rounding mode; supports all rounding modes described in the *Datapath Floating-Point Overview*
z	(*sig_width* + *exp_width* + 1) bits	Output	(`a * b`) + (`c * d`) + (`e * f`)

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
status	8 bits	Output	Status flags corresponding to `z`; for details, see STATUS Flags in the *Datapath Floating-Point Overview*

Table 1-2 Parameter Description

Parameter	Values	Description
sig_width	2 to 253 bits	Word length of fraction field of floating-point numbers `a`, `b`, `c`, `d`, `e`, `f`, and `z`
exp_width	3 to 31 bits	Word length of biased exponent of floating-point numbers `a`, `b`, `c`, `d`, `e`, `f`, and `z`
ieee_compliance	0 or 1	Level of support for IEEE 754: ■ 0: No support for IEEE 754 NaNs and denormals; NaNs are considered infinities and denormals are considered zeros ■ 1: Fully compliant with the IEEE 754 standard, including support for NaNs and denormals For more, see IEEE 754 Compatibility in the *Datapath Floating-Point Overview*.
arch_type	0 or 1	Controls the use of an alternative architecture Default: 0 (previous architecture)

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW02.DW_FP_DP3_CFG_SIM	Design unit name for VHDL simulation
dw/dw02/src/DW_fp_dp3_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_fp_dp3.v	Verilog simulation model source code

Table 1-5 Functional Description

a	b	c	d	e	f	status	z[a]
a (FP)	b (FP)	c (FP)	d (FP)	e (FP)	f (FP)	*	a*b + c*d + e*f (FP)

a. The actual value of the result is defined by the rounding mode.

Parameters *ieee_compliance* and *arch_type* control the functionality of this component. Different values of *arch_type* result in slightly different numeric behaviors, but the component is more accurate than the implementation of the same function using basic floating-point multipliers and adders.

When the parameter *arch_type* is set to 0 the component calculates the dot-product function as if the operation was done using infinite precision followed by a single rounding step at the end to generate the final result.

When *arch_type* = 1, a special architecture is used to reduce hardware and the component behaves similar to a network of independent FP operators (network of multipliers and adders), without intermediate rounding. Only one rounding step is performed to compute the output value. In this case, the component produces logic that is not as accurate as when *arch_type* = 0, but it is still more accurate than the network of multipliers and adders. When *arch_type* = 1, the logic created by the component is smaller and faster than when *arch_type* = 0.

Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If an invalid rounding mode has been detected on `rnd`, the following message is displayed:

  ```
  WARNING: <instance_path>:
      at time = <timestamp>: Illegal rounding mode.
  ```

  To suppress this message, use the DW_SUPPRESS_WARN macro explained earlier.

Related Topics

- Datapath Floating-Point Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DW_Foundation_comp_arith.all;

entity DW_fp_dp3_inst is
      generic (
        inst_sig_width : POSITIVE := 23;
        inst_exp_width : POSITIVE := 8;
        inst_ieee_compliance : INTEGER := 0;
        inst_arch_type : INTEGER := 0
        );
      port (
        inst_a : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_b : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_c : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_d : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_e : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_f : in std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        inst_rnd : in std_logic_vector(2 downto 0);
        z_inst : out std_logic_vector(inst_sig_width+inst_exp_width downto 0);
        status_inst : out std_logic_vector(7 downto 0)
        );
    end DW_fp_dp3_inst;


architecture inst of DW_fp_dp3_inst is

begin

    -- Instance of DW_fp_dp3
    U1 : DW_fp_dp3
    generic map (
          sig_width => inst_sig_width,
          exp_width => inst_exp_width,
          ieee_compliance => inst_ieee_compliance,
          arch_type => inst_arch_type
          )
    port map (
          a => inst_a,
          b => inst_b,
          c => inst_c,
          d => inst_d,
          e => inst_e,
          f => inst_f,
          rnd => inst_rnd,
          z => z_inst,
          status => status_inst
          );
```

```
end inst;

-- pragma translate_off
 configuration DW_fp_dp3_inst_cfg_inst of DW_fp_dp3_inst is
 for inst
 end for; -- inst
 end DW_fp_dp3_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_fp_dp3_inst( inst_a, inst_b, inst_c, inst_d, inst_e,
          inst_f, inst_rnd, z_inst, status_inst );

parameter inst_sig_width = 23;
parameter inst_exp_width = 8;
parameter inst_ieee_compliance = 0;
parameter inst_arch_type = 0;


input [inst_sig_width+inst_exp_width : 0] inst_a;
input [inst_sig_width+inst_exp_width : 0] inst_b;
input [inst_sig_width+inst_exp_width : 0] inst_c;
input [inst_sig_width+inst_exp_width : 0] inst_d;
input [inst_sig_width+inst_exp_width : 0] inst_e;
input [inst_sig_width+inst_exp_width : 0] inst_f;
input [2 : 0] inst_rnd;
output [inst_sig_width+inst_exp_width : 0] z_inst;
output [7 : 0] status_inst;

    // Instance of DW_fp_dp3
    DW_fp_dp3 #(inst_sig_width, inst_exp_width, inst_ieee_compliance, inst_arch_type)
U1 (
                .a(inst_a),
                .b(inst_b),
                .c(inst_c),
                .d(inst_d),
                .e(inst_e),
                .f(inst_f),
                .rnd(inst_rnd),
                .z(z_inst),
                .status(status_inst) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2020	DWBB_201912.5	■ Adjusted the description of the *ieee_compliance* parameter in Table 1-2 on page 2 ■ Added "Suppressing Warning Messages During Verilog Simulation" on page 3 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com